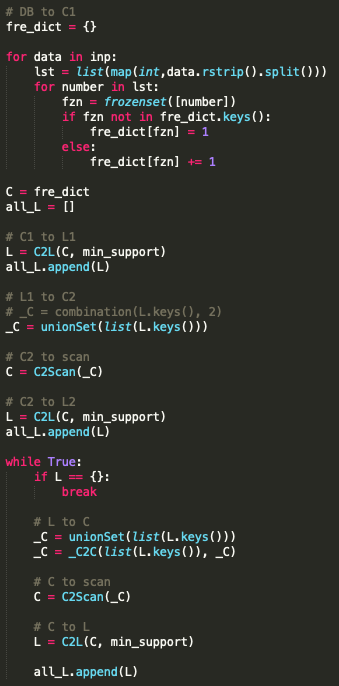
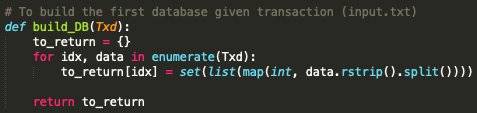
R07922141

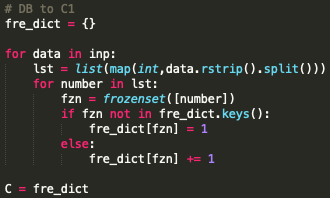
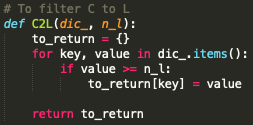
張緣彩

18/10/2018

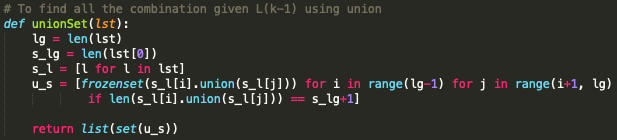
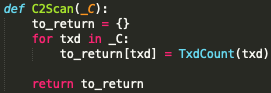
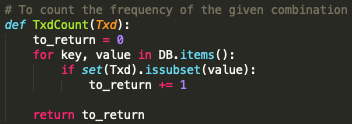
Data Mining Homework 1 Report

1. Implementation
   1. Apriori  
      My main code fragment is as the figure below, I will describe in detail in bottom.  
        
      First, we read the data and build a database. The database is dictionary type data structure with transaction id as key and a set of itemset as value.  
      

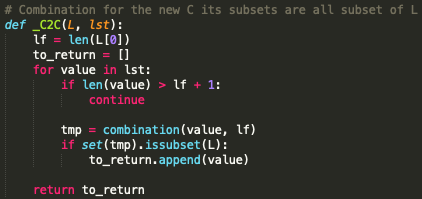
Second, we iterate the all the transactions to generate C1 (all the items) and count the support of each item. Then from C1 we filter out those items that are not greater than min support (function C2L) to generate L1.

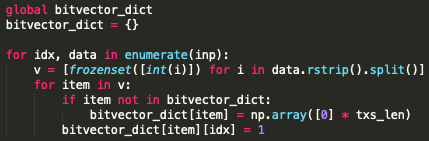
From L1 to generate C2 by combine any 2 items in L1 (function unionSet). For example,   
a. if L2 = {AB, AC, CE} as input then unionSet will output C3 = {ABC, ACE}. (ABCE will be drop)  
b. if L1 = {A, B, C, E} as input then unionSet will output C2 = {AB, AC, AE, BC, BE, CE}.  
We don’t need to check if subsets of each item in C2 are the subset of L1.

  
After generate the candidate C2, we need to scan through database and filter out those itemsets that are not greater than min support (function C2Scan, C2L) and generate L2.  
 

From L2 we generate C3 using the same technique. But before we scan through the database, we have to filter out the itemsets in C3 which its subsets are not in L2 (function \_C2C). For example,  
a. L2 = {AB, AC, BC, BE}, Temp C3 = {ABC, ABE} then we filter out ABE since its subset {AE} is not in L2. So C3 = {ABC}.



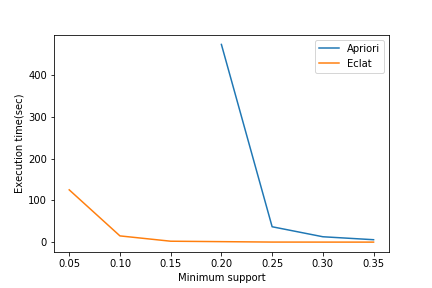
After generate C3 we can scan through the database to generate L4. This process will keep iterate until no new L is generate.

* 1. Eclat  
     My main code fragment for eclat algorithm is as figure below. Since I use recursive depth first search, the code will be short compare to apriori. I will also describe in detail in bottom.  
       
     First, we read the data and build a vertical bit vector database.  
       
     Second, apply Eclat recursive depth first search function. Given an empty set as root.  
       
     The function iterates all the itemsets. The arguments of the function are current itemset and the candidate itemsets. For example, if the itemsets we have are {A, B, C, D, E}, for current = A then the candidates of A will be {B, C, D, E}; for current = B then the candidates of B will be {C, D, E} since {AB} will be generate when current = A so that B does not need A as its candidate. For every recursive call, we will iterate the candidates and calculate the support of the new itemset (union of current and the candidate’s item). If the support of the new itemset is greater than minimum support, the new itemset will become the next current and the candidate’s item will be saved in the next candidates list. For performance reason, the new itemset bit vector will also be saved in the bit vector dictionary. After the iteration, we will apply the function to each next current and next candidate recursively until there is no new next candidate generated.  
     I found out that if I use ‘&’ instead of np.logical\_and() as logic AND, it will hurt the performance. And if I use sum() or np.sum() instead of np.count\_nonzero() as to add all the intersected bit vector, it will hurt the performance really bad.

1. Graph

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.05 | 0.1 | 0.15 | 0.2 | 0.25 | 0.3 | 0.35 |
| Apriori | - | - | - | 473.90 | 36.86 | 12.88 | 5.78 |
| Eclat | 125.324 | 14.9295 | 2.1841 | 1.1528 | 0.13 | 0.0624 | 0.0247 |

Plot in seconds



Plot in seconds (Log-scaled)

